Professional paper
Accepted 4. 12. 2006.

GÜNTER WALLNER

Geometry of Real Time Shadows

Geometry of Real Time Shadows

ABSTRACT

Shadows provide important visual hints about the spatial relationship between objects. Shadow volumes are one way to generate sophisticated shadows for use in real time environments. This paper focuses on the geometric aspects which are involved in the creation of the shadow volume. Speed up techniques like shaders and dual space approaches for silhouette determination are discussed. Finally the application of the described methods in a software for shadow profile calculation is explained.

Key words: Shadow volumes, Dual space, Silhouette determination, Shader, Real time

MSC 2000: 51-04

Geometrija sjenâ u realnom vremenu

SAŽETAK

Sjene pružaju važne vizualne informacije o prostornom odnosu među objektima. Tijelo sjene je jedan način kako generirati profinjene sjene za prkaze u realnom vremenu. U ovom članku usredotočilo se na geometrijski aspekt uključen u stvaranje tijela sjene. Razmatraju se brze i efikasne tehnike za određivanje rastavnice preko dva pristupa: dualnog prostora i programa za sjenčanje (shadera). Na kraju je prikazana primjena opisanih metoda u softveru za određivanje oblika sjene.

Ključne riječi: tijelo sjene, dualni prostor, određivanje rastavnice, program za sjenčanje, realno vrijeme

1 Introduction

Shadows are an important part in computer graphics because they can reveal information that otherwise would not be ascertainable. Foremost, they reveal the spatial relationship between objects in the scene. They also disclose new angles on an object that otherwise might not be visible and they can also indicate the presence of off-screen objects. These and other visual functions of shadows in computer graphics are described by Birn in [5].

Shadow volumes were first proposed by Crow in 1977 [8]. With the advent of modern day computer graphic cards, shadow volumes are now possible in real time. Heidmann [14] adapted Crow's algorithm to hardware acceleration. His method is now known as the z-pass method (because the stencil buffer is incremented/decremented when a polygon passes the depth test). However, the z-pass method does not work correctly if the near clipping plane intersects the shadow volume. Carmack [6] solved the problem by using z-fail testing (the stencil buffer is incremented/decremented when a polygon fails the depth test). The z-fail method still yields incorrect results if the shadow volume is intersected by the far clipping plane. This problem can be circumvented by moving the far clipping plane to infinity, as proposed by [9].

Shadow maps (introduced by [26]) are image based alternatives to shadow volumes (which operate on the object geometry). In the meantime several different shadow map algorithms have been developed. Both methods have their benefits and drawbacks. For a comparison of the pros and cons of both methods see for example [25].

"Classic" shadow volume algorithms create hard shadows. A shadow region is divided into two parts: the region which is fully in shadow (umbra) and the region which is partially in shadow (penumbra). Hard shadows only consist of the umbra area. Soft shadow volume algorithms have been developed among others by Ulf Assarsson and Tomas Akenine-Möller [21, 1].

2 Assumptions and Definitions

The shadow volume algorithm requires that the shadow casting objects must be a 2-manifold polygon mesh and free of non-planar polygons. 2-manifold means that every edge of the mesh must be shared exactly by two polygons. It is also useful to restrict oneself to triangular meshes, because modern graphics hardware is optimized for triangle rendering.

Furthermore, all triangles must have the same winding order. For the following discussion a counter clockwise winding order and outward pointing normals are assumed.

A silhouette edge is an edge adjacent to one front-facing and one back-facing polygon. A polygon is called front-

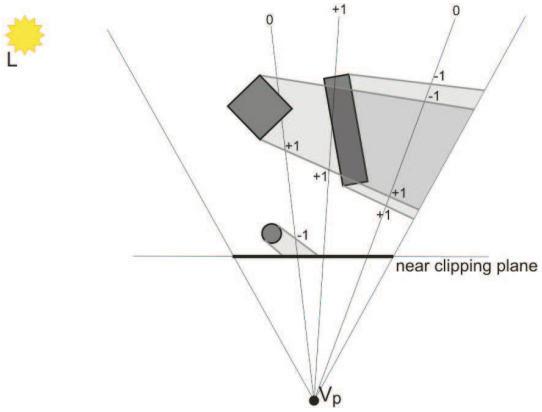

Figure 1: *The z-pass method. The values at the end of the rays represent the values left in the stencil buffer. Note that the stencil value of the leftmost ray is wrong due to the clipping of the shadow volume of the sphere at the near clipping plane.*

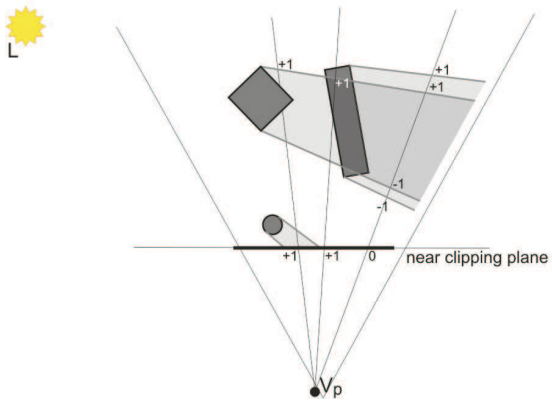

Figure 2: *The z-fail method. The values at the intersection of the ray and the near clipping plane represent the values left in the stencil buffer. This time the stencil value for the ray passing through the sphere is correct.*

facing with respect to the light if the dot-product of its normal and the vector from the light position and a point on the polygon is positive. Respectively a polygon is called back-facing with respect to the light if the dot-product is negative.

A border edge is an edge which is only adjacent to one face (which implies that the mesh is open). It should be noted that we can handle open meshes if we treat border edges as part of the silhouette. The silhouette is the set of all silhouette edges (and border edges).

3 Overview

I will first give an overview of the z-pass algorithm and then point out the differences with respect to the z-fail algorithm. The basic concept [...] is to use the stencil buffer as a masking mechanism to prevent pixels in shadow from being drawn during the render pass for a particular light source [17]. First of all the stencil buffer is initialized with zero and the z-buffer is initialized with the depth values of the visible objects during a first rendering pass. In this pass only light-independent attributes are considered (e.g. ambient light). Then the shadow volume is rendered with writes to the color buffer and depth buffer disabled. This is usually done in two steps. First, the front faces of the shadow volume (with respect to the camera position) are rendered and the stencil buffer is incremented each time the fragment passes the depth test. Second, the back faces are rendered. This time decrementing the value in the stencil buffer when a fragment passes the depth test.

As shown in figure 1, this leaves non-zero values in the stencil buffer wherever the shadow volume intersects a visible object. Figure 1 in addition shows why this approach fails if the shadow volume intersects the near clipping plane.

As noted by [3] the front faces must be rendered before the back facing polygons to avoid shadow counting overflow. That is, because under OpenGL the result of the increment and decrement functions is clamped to lie between 0 and the maximum unsigned integer value ($2^n - 1$ if the stencil buffer holds n bits) [22]. However, rendering the shadow volume geometry twice is a suboptimal solution. The OpenGL extension EXT_stencil_two_side [11] allows separate stencil states for front faces and back faces to be specified simultaneously. Therefore front faces as well as back faces can be rendered at once. Though this time it is not guaranteed that the front facing polygons will be rendered before the back faces. Consequently the feasibility exists that the stencil value for a particular pixel is decremented before it is incremented. We can account for that option by using another OpenGL extension, namely EXT_stencil_wrap [12], which allows stencil values to wrap when they exceed the maximum and minimum stencil values.

Several authors [4, 3, 6] proposed methods to cap the shadow volume at the near plane. However, these are computationally expensive and they have robustness problems.

Carmack [6] and others therefore suggested the z-fail algorithm. Instead of counting the shadow faces in front of a particular pixel, the shadow faces behind are counted. This time the near clipping plane problem is avoided because shadow volume geometry between the eye and the pixel is nonrelevant. Figure 2 shows the z-fail approach. As already mentioned in the introduction the z-fail approach

moves the near clipping plane problem to the far plane, which can be prohibited by using an infinite projection matrix (see section 6).

4 Silhouette Detection

To calculate the shadow volume, we first have to determine the silhouette of the shadow casting object. The so-called brute force method for detecting silhouette edges is to loop through all edges and check the dot-product of the adjacent triangles. Since silhouette detection is one of the two major bottlenecks (beside fill rate consumption), as pointed out by [16], it is appropriate to use more sophisticated methods. [2] developed a dual space approach for silhouette extraction in 3D and [15] used a similar method but moved to four dimensions. Most recently [24] presented a paper about silhouette extraction in Hough space.

Because [15] are concerned with non photorealastic rendering, they determine the silhouette with respect to the viewpoint. However, in case of shadows the silhouette depends on the light position. Therefore the viewpoint must be substituted with the light position. The algorithm in [15] is based on the geometric concept of duality in a projective space and the following characterization of the silhouette: If $\mathbf{L}$ is the homogeneous light position, the set of silhouette points determines a general cone C (with apex $\mathbf{L}$) tangent to the differentiable surface M. If $\mathbf{L}'$ is the image plane of $\mathbf{L}$ when applying the duality map, the image C' of C is the intersection of the plane $\mathbf{L}'$ with the dual surface M'. C' can be identified with the silhouette set of surface M. A point $\mathbf{v} = (v_x, v_y, v_z, 1)$ of M belongs to the silhouette set if $(\mathbf{a} - \mathbf{v}) \cdot \mathbf{n} = 0$, where $\mathbf{n} = (n_x, n_y, n_z, 0)$ is the unit normal vector to M at $\mathbf{v}$ and $\mathbf{a} = (a_x, a_y, a_z, 1)$ is a point on the tangent plane at $\mathbf{v}$. The tangent plane at $\mathbf{v}$ is mapped onto a dual point $\mathbf{v}' = (v_x, v_y, v_z, -\mathbf{v} \cdot \mathbf{n})$. Therefore the silhouette set of M is characterized by the equation $\mathbf{L} \cdot \mathbf{v}' = (\mathbf{L} - \mathbf{v}) \cdot \mathbf{n} = 0$. Consequently, the problem of finding the silhouette of a differentiable surface is reduced to the problem of intersecting a plane with a surface.

Since I am concerned with polyhedral surfaces the problem can be reformulated in a way as described by [15]. The dual surface is built by mapping each vertex $\mathbf{v}$ of the mesh onto a homogeneous point $\mathbf{v}' = (v_x, v_y, v_z, -(\mathbf{v} \cdot \mathbf{n}))$. The dual surface has the same connectivity but different vertex positions. A dual edge e' of an edge $e = (\mathbf{v_1}, \mathbf{v_2})$ is a tuple $(\mathbf{v_1'}, \mathbf{v_2'})$. An edge e belongs to the set of silhouette edges if $\mathbf{L} \cdot \mathbf{v_1'} >= 0$ and $\mathbf{L} \cdot \mathbf{v_2'} < 0$ or vice versa. Each $\mathbf{v}'$ is then normalized (using the Euclidean norm) to make sure that each point of the dual surface lies inside the unit hypercube. This allows us to store each dual edge in a 4D variant of an octtree (I will call it hextree in the further discussion) as pointed out by [7]. At the highest level this hextree ranges from $(-1,-1,-1,-1)$ to $(1,1,1,1)$. The space can be repeatedly divided into 16 smaller hextrees until a small enough partition is reached. A dual edge e' is then inserted into the smallest subcube which encloses $\mathbf{v_1'}$ as well as $\mathbf{v_2'}$.

Instead of using two bounding boxes per subcube to determine if the dual edges have to be verified [7] I use a different approach. For testing if an AABB[1] and a plane intersect, the box diagonal which is most aligned with the normal of the plane has to be found first. Second the diagonals vertices ($\mathbf{v_{min}}$ and $\mathbf{v_{max}}$) are inserted into the plane equation. If the signs of the results differ or at least one of them is zero, then the plane intersects the box [20]. [20] also points out that the two vertices can be found directly. The signs of the components of the plane normal are used as a bit mask. If this mask is interpreted as a number it can be used as index to an array of AABB vertices. This approach can easily be extended to four dimensions. Each of the 16 vertices of a 4D cube is stored in an array so that the minimum vertex is located at index 0 and the maximum vertex at position 15. Instead of the plane normal we interpret the signs of the components of $\mathbf{L}$ as a bit mask. The index i of $\mathbf{v_{min}}$ can then be calculated as $i = 8 \cdot sgn(L_x) + 4 \cdot sgn(L_y) + 2 \cdot sgn(L_z) + sgn(L_h)$ where

$$sgn(x) = \begin{cases} 0 & x >= 0 \\ 1 & \text{otherwise.} \end{cases}$$

The $\mathbf{v_{max}}$ vertex can be found by inverting the bit mask. The dual edges of a subcube must only be tested if $\mathbf{L} \cdot \mathbf{v_{min}} >= 0$ and $\mathbf{L} \cdot \mathbf{v_{max}} < 0$ or vice versa.

Building the dual surface and inserting the dual edges into the hextree can be done once in a preprocessing step as long as the connectivity of the object does not change. Furthermore silhouette detection must only be performed if the object position changes with respect to the light position.

5 Shadow Volume Construction

Once the set of silhouette edges is determined the edges must be extruded to form the shadow volume. As described by [17], no matter what finite distance silhouette edges are extruded, it is still possible that the shadow volume does not reach far enough to cast a shadow on every object in the scene that should intersect the volume. This problem worsens when the light source is very near to the shadow casting object, but it can be circumvented by using an infinite projection matrix. How this matrix can be obtained is described in section 6.

[1] AABB stands for Axis Aligned Bounding Box. Assuming an AABB is valid in our case because the hextree is axis aligned.

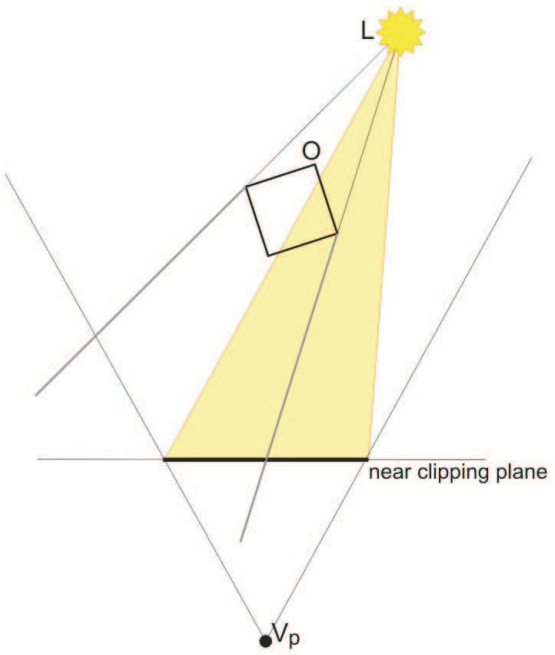

Figure 3: *An Object O is casting a shadow onto the near clip plane since it partially intersects the near clip volume (shaded)*

To make the z-fail algorithm work correctly, the shadow volume must be a closed volume where all polygons must have a consistent winding order. A complete shadow volume consists of: (1) the front cap (consisting of all front-facing polygons), (2) the extruded silhouette edges and (3) the back cap. It is notable that the extrusion of the geometry depends on the light source. For a point light the vertices of the silhouette edge must be extruded to infinity along the vector from the location of the point light to the vertex (see figure 4). If $\mathbf{v} = (v_x, v_y, v_z, 1)$ is the position of the vertex to be extruded and $\mathbf{L}$ is the position of the point light then the extruded vertex $\mathbf{v}_e = (v_x - L_x, v_y - L_y, v_z - L_z, 0)$.

For a directional light all extruded points converge to a single point in infinity (see figure 4) at position $(-L_x, -L_y, -L_z, 0)$. This implies that the back cap is not necessary for directional light sources. The back cap conventionally consisted of all back-facing polygons projected away from the light [9, 17]. But since the back cap is at infinity the shape does not matter [19]. The only constrain which remains is that the back cap must actually close the volume. This can be achieved with a simple triangle fan constructed from the extruded silhouette edges [19, 16].

The z-pass algorithm doesn't use caps, therefore the incorrect results if the shadow volume intersects the near clip plane (see [9] for details) or the viewpoint is inside the volume. From this point it is clear that the z-fail method is computationally more expensive and should only be used when necessary. To determine whether the shadow volume is clipped by the near plane the near clip volume has to be constructed. The near clip volume is bound by the planes which connect the near rectangle to the light position, as shown in figure 3. The near rectangle is the area cut out of the near plane by the four side planes of the view frustum. Only an object which is inside this near clip volume can cast a shadow onto the near clipping plane. For a comprehensive description see [17].

Silhouette edge extrusion can now be done on graphics hardware to remove the burden from the CPU. The following Cg vertex shader extrudes a vertex $\mathbf{v} = (v_x, v_y, v_z, v_w)$ if $v_w = 0$ otherwise the position is just passed through.

```
float4 lightToVertex = IN.position -
    lightPos;

float m = 1 - IN.position.w;
float4 outx = IN.position*(1-m) +
              lightToVertex*m;
outx.w = IN.position.w;

// transform position to homogeneous clip
    space
OUT.HPOS = mul(ModelViewProj, outx);
```

IN. position is the vertex coordinate and lightPos is the position of the point light. If shaders are used, one has to take care of transforming the vertex position into homogenous clip space, therefore the multiplication with the modelview-projection matrix. To make this approach work correctly, each vertex of the silhouette must be passed twice to the shader. Once with $v_w = 1$ and once with $v_w = 0$. The extrusion for a directional light looks similar.

6 Infinite Projection Matrix

The OpenGL projection matrix is defined as [22]:

$$P = \begin{vmatrix} \frac{2\cdot n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2\cdot n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2\cdot f\cdot n}{f-n} \\ 0 & 0 & -1 & 0 \end{vmatrix}$$

In this matrix f is the distance from the viewer to the far clip plane, n the distance to the near clip plane and r and l are the respective distances to the left and right clip plane. t and b are the distances to the top and bottom clip plane.

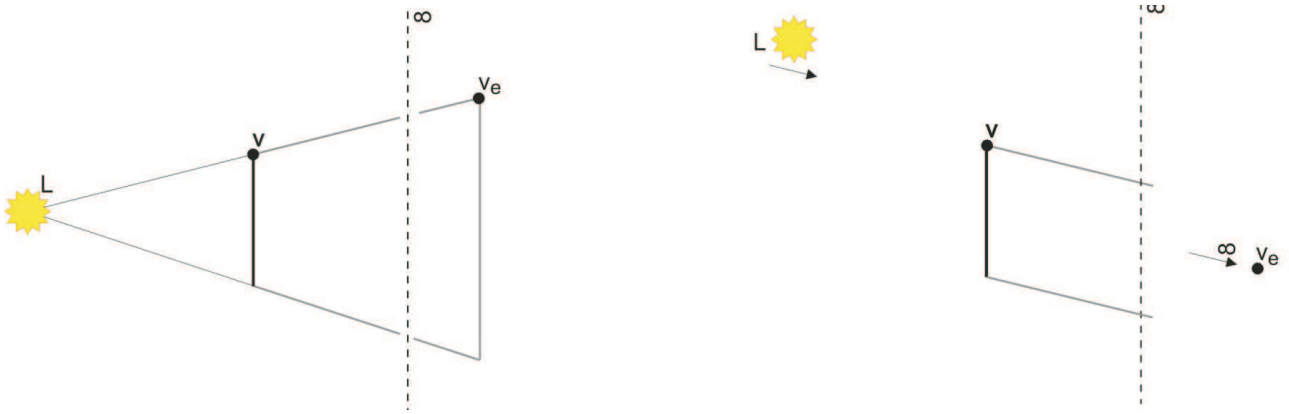

Figure 4: *Silhouette extrusion for a point light (left) and for a directional light (right)*

We can obtain the infinite projection matrix by calculating $P_\infty = \lim_{f\to\infty} P$ which yields

$$P_\infty = \begin{vmatrix} \frac{2\cdot n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2\cdot n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -1 & -2\cdot n \\ 0 & 0 & -1 & 0 \end{vmatrix}$$

An infinite projection matrix reduces the depth buffer precision only marginally as pointed out by [9]. However, if you are concerned about this loss you can use Nvidia's NV_depth_clamp [23] extension. If depth clamping is enabled, the near and far clipping plane are disabled for rasterizating geometry primitives.

7 Rendering

Here I present the necessary steps to render shadow volumes with OpenGL. First we render the scene with enabled depth writes, backface culling and with ambient lighting only (light independent attributes). This guarantees that the depth buffer is initialized with the correct depth values. Afterwards we disable writes to the depth buffer and turn off ambient lighting.

```
glEnable(GL_LIGHTING);
glLightModelfv(GL_LIGHT_MODEL_AMBIENT,
    ambient);
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LESS);
glEnable(GL_CULL_FACE);
glCullFace(GL_BACK);

drawScene();

glDepthMask(GL_FALSE);
glLightModelfv(GL_LIGHT_MODEL_AMBIENT,
    zero);
```

The stencil mask has to be calculated separately for each light source.

```
for each light source
{
```

First we clean the stencil buffer, configure the stencil test so that it always passes and disable writes to the color buffer. We will take advantage of two side stencil testing so that we only have to render the shadow volume of each occluder once. Therefore the stencil operation is set to increment and decrement for front- and back-facing polygons respectively – if the depth test fails. Culling is also turned off because front as well as back faces must be rendered at the same time.

```
  glClear(GL_STENCIL_BUFFER_BIT);
  glEnable(GL_STENCIL_TEST);
  glStencilFunc(GL_ALWAYS, 0, ~0);
  glStencilMask(~0);

  glColorMask(GL_FALSE, GL_FALSE, GL_FALSE,
      GL_FALSE);

  glActiveStencilFaceEXT(GL_BACK);
  glStencilOp(GL_KEEP, GL_INCR_WRAP_EXT,
      GL_KEEP);
  glActiveStencilFaceEXT(GL_FRONT);
  glStencilOp(GL_KEEP, GL_DECR_WRAP_EXT,
      GL_KEEP);

  glDisable(GL_CULL_FACE);
  glEnable(GL_STENCIL_TEST_TWO_SIDE_EXT);
```

Now the shadow volume of each occluder in the scene is rendered. Afterwards culling is turned on and the stencil test is disabled. At that time the stencil buffer holds the correct information about which pixels are in shadow and which aren't.

```
  for each occluder
  {
    renderShadowVolume(occluder);
  }

  glEnable(GL_CULL_FACE);
  glDisable(GL_STENCIL_TEST_TWO_SIDE_EXT);
```

The whole scene is now rendered again. This time the current light is enabled and configured (all light dependent attributes). Stencil testing is configured so that only pixels with a zero stencil value are rendered. Equal depth testing is used so that only visible fragments are updated. Since this pass adds to the ambient scene already in the color buffer, additive blending must be enabled as well as writes to the color buffer. After rendering the scene, blending is disabled and the depth function is restored to less depth testing.

```
  glEnable(light);
  configureLight(light);

  glEnable(GL_BLEND);
  glBlendFunc(GL_ONE, GL_ONE);
  glColorMask(GL_TRUE, GL_TRUE, GL_TRUE,
      GL_TRUE);

  glStencilFunc(GL_EQUAL, 0, ~0);
  glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
  glDepthFunc(GL_EQUAL);

  renderScene();

  glDisable(GL_BLEND);
  glDepthFunc(GL_LESS);
}
```

After the above steps have been carried out for all lights, stencil testing is disabled and writes to the depth buffer are enabled.

```
glDisable(GL_STENCIL_TEST);
glDepthMask(GL_TRUE);
```

8 Application: Shadow Profiles

I have succesfully applied shadow volumes in an application for calculating shadow profiles in real time. A shadow profile shows the cast shadow of an object over a specific time period. This is, for example, of concern for architects to find out how long the surroundings are obscured by a building. After providing the required information needed for computing the position of the sun[2] (latitude, date, time) and the time period, the shadow profile is calculated.

[2] See [13] for a description of the calculation

Scene	Number of triangles[a]	∼time [ms]
Eiffel Tower	11353 (11155)	4.584
Industry Area	13615 (13585)	7.299
Uniqua Building	182038 (147296)	44.486
Uniqua Building	182038 (182038)	58.666

Table 1: Performance with brute force silhouette detection

[a] First number: total triangles in the scene. Second number: triangles of shadow casting objects

Scene	Number of triangles	∼time [ms]
Eiffel Tower	11353 (11155)	4.236
Industry Area	13615 (13585)	5.799
Uniqua Building	182038 (147296)	39.331
Uniqua Building	182038 (182038)	48.872

Table 2: Performance with dual space silhouette detection

The application can detect the silhouette either by brute force or with the above described dual space approach. If the graphics card supports vertex and fragment shaders, silhouette extrusion and per pixel lighting is performed on the GPU. Otherwise the CPU handles the extrusion and standard OpenGL lighting is used. Double sided stencil testing is performed if EXT_stencil_two_side is supported. The z-fail algorithm is only applied if necessary (see section 3).

Figures 5 to 7 show some sample scenes. Table 1 shows the time needed for brute force silhouette detection for each scene and table 2 for dual space silhouette detection, respectively. All measurements were taken on a Pentium 4 3.4Ghz processor with 1GB memory. For each scene a hextree with a fixed depth of four was chosen for the dual space approach.

9 Future Work

The results show that silhouette detection can greatly improve performance. As future work it would be interesting to see how Hough space silhouette finding [24] can further speed up the process. At this time no techniques to reduce fill rate consumption are implemented. Lengyel [17] describes how OpenGLs scissor rectangle support can be used to cut down the fill rate penalty for rendering the shadow volumes. That is because the hardware does not generate fragments outside the scissor rectangle. The scissor rectangle can be applied on a per light basis or per geometry basis, as pointed out by [18]. [10] suggest a depth bounds test for stencil writes. This idea is based on the observation that some depth values can never be in shadow, so incrementing and decrementing the stencil buffer is needless.

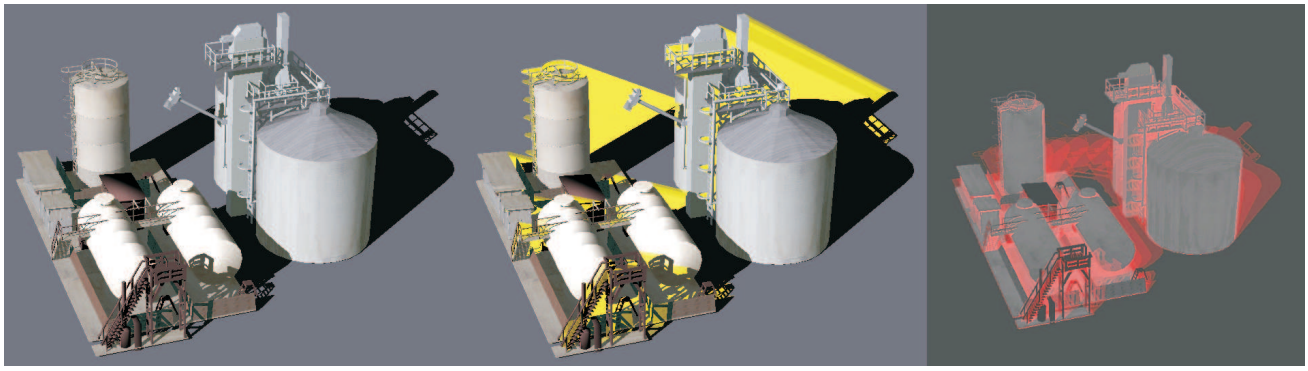

Figure 5: *left: Casted shadows of an industry area located at a latitude of 45.2° north on September 18th at 3pm. middle: Visualization of the shadow volumes (yellow). right: The shadow profile of the scene over a time period of three hours (12pm until 3pm in 30 minutes time steps).*

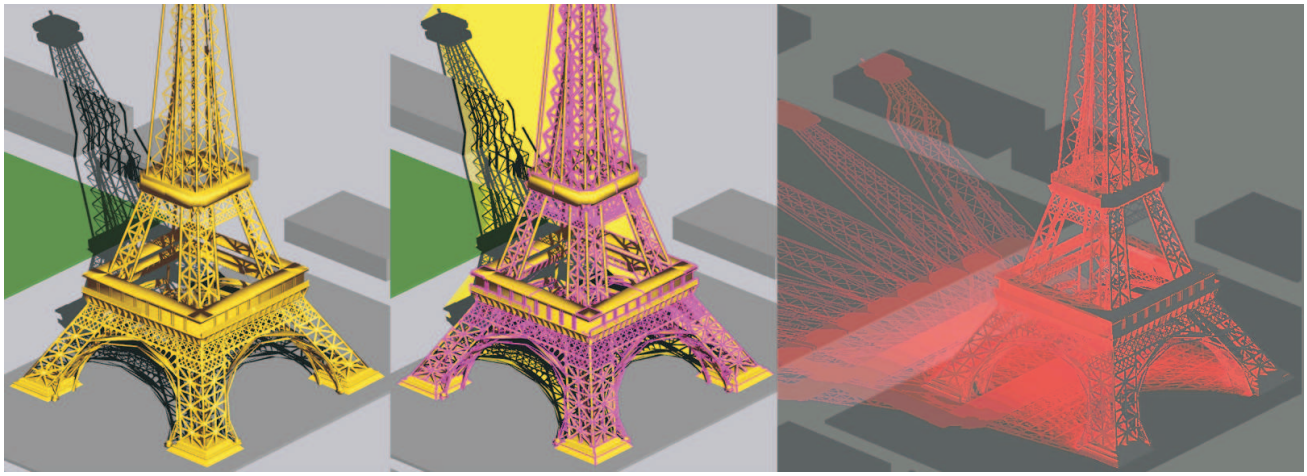

Figure 6: *left: Shadow of the Eiffel Tower in Paris (latitude of 48.8° north) on September 18th at 2pm. middle: Visualization of the shadow volume (yellow) and the silhouette edges (pink). right: Shadow profile over a time period of four hours (10am until 2pm in 30 minutes intervals).*

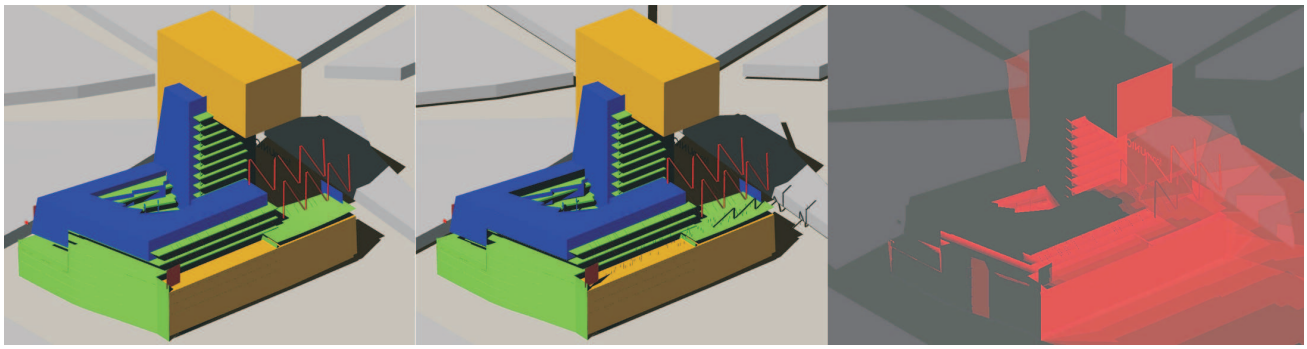

Figure 7: *Proposal for the Uniqua building in Vienna (48.2° north) by Hans Hollein. A color was assigned to each structural component. left: Only the facade (yellow) and the concrete (green) is casting a shadow. middle: The complete building is casting a shadow. right: The shadow profile over a time period of eight hours (9am until 5pm in 1 hour intervals) on September 18th.*

10 Conclusions

In this paper I have presented the necessary steps for a robust implementation of stencil shadow volumes. Stencil shadow volumes suffer mainly from two bottlenecks: (a) fill rate and (b) silhouette detection. The latter was discussed in section 4. Modern graphics hardware can take over computations which formerly had to be performed on the CPU, e.g. silhouette extraction. Code snippets showed how stencil shadows can be implemented with OpenGL. Extensions to OpenGL provide further ways to improve performance.

References

[1] ULF ASSARSSON AND TOMAS AKENINE-MÖLLER, *A Geometry-based Soft Shadow Volume Algorithm using Graphics Hardware*, Siggraph Proceedings **22**, 511–520, 2003, available online: `http://www.cs.lth.se/home/Tomas_Akenine_Moller/pubs/soft_sig2003.pdf`

[2] G. BAREQUET AND C. A. DUNCAN AND M. T. GOODRICH AND S. KUMAR AND M. POP, *Efficient perspective-accurate silhouette computation*, Proceedings of the 15th annual symposium on Computational geometry, 417–418, 1999

[3] HARLEN COSTA BATAGELO AND ILAIM COSTA JUNIOR, *Real-Time Shadow Generation using BSP Trees and Stencil Buffers*, XII Brazilian Symposium on Computer Graphics and Image Processing, 93–102, 1999

[4] JASON BESTIMT AND BRYANT FREITAG, *Real-Time Shadow Casting Using Shadow Volumes*, 1999, available online: `http://www.gamasutra.com/features/19991115/bestimt_freitag_01.htm`

[5] JEREMY BIRN, *Digital Lighting & Rendering*, New Riders, 2006

[6] JOHN CARMACK, *E-Mail to private list*, 2000, available online: `http://developer.nvidia.com/attach/6832`

[7] JOHAN CLAES AND FABIAN DI FIORE AND GERT VANSICHEM AND FRANK VAN REET, *Fast 3D Cartoon Rendering with Improved Quality by Exploiting Graphics Hardware*, Proceedings of Image and Vision Computing New Zealand (IVCNZ), 13–18, 2001, available online: `http://www.cs.utah.edu/npr/papers/Claes_IVCNZ2001.pdf`

[8] F.C. CROW, *Shadow Algorithms for Computer Graphics*, Siggraph Proceedings **11**, 242–248, 1977

[9] CASS EVERITT AND MARK J. KILGARD, *Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering*, 2002, available online: `http://developer.nvidia.com/object/robust_shadow_volumes.html`

[10] CASS EVERITT AND MARK J. KILGARD, *Optimized Stencil Shadow Volumes*, Game Developer Conference Presentation, 2003, available online: `http://developer.nvidia.com/object/GDC_2003_Presentations.html`

[11] OPENGL EXTENSION REGISTRY, *EXT_stencil_two_side*, 2003, available online: `http://oss.sgi.com/projects/ogl-sample/registry/EXT/stencil_two_side.txt`

[12] OPENGL EXTENSION REGISTRY, *EXT_stencil_wrap*, 2002, available online: `http://oss.sgi.com/projects/ogl-sample/registry/EXT/stencil_wrap.txt`

[13] GEORG GLAESER, *Der mathematische Werkzeugkasten*, Elsevier, 2006

[14] TIM HEIDMANN, *Real Shadows, Real Time*, Silicon Graphics Inc. **18**, 23–31, 1991

[15] AARON HERTZMANN AND DENIS ZORIN, *Illustrating smooth surfaces*, Proceedings of the 27th annual conference on Computer graphics and interactive techniques, 517–526, 2000, available online: `http://mrl.nyu.edu/publications/illustrating-smooth/hertzmann-zorin.pdf`

[16] HUN YEN KWOON, *The Theory of Stencil Shadow Volumes*, available online: `http://www.gamedev.net/reference/articles/article1873.asp`

[17] ERIC LENGYEL, *The Mechanics of Robust Stencil Shadows*, 2002, available online: `http://www.gamasutra.com/features/20021011/lengyel_01.htm`

[18] ERIC LENGYEL, *Advanced Stencil Shadow and Penumbral Wedge Rendering*, Game Developer Conference Presentation, 2005, available online: `http://www.terathon.com/gdc_lengyel.ppt`

[19] MORGAN MCGUIRE AND JOHN F. HUGHES AND KEVIN T. EGAN AND MARK J. KILGARD AND CASS EVERITT, *Fast, Practical and Robust Shadows*, 2003, available online: `http://developer.nvidia.com/object/fast_shadow_volumes.html`

[20] TOMAS MÖLLER AND ERIC HAINES, *Real-Time Rendering*, A K Peters, 1999

[21] TOMAS AKENINE-MÖLLER AND ULF ASSARSSON, *Approximate Soft Shadows on Arbitrary Surfaces using Penumbra Wedges*, Proceedings of the 13th Eurographics workshop on Rendering, 297–306, 2002, available online: http://www.ce.chalmers.se/~uffe/softshadows_egrw.pdf

[22] JACKIE NEIDER AND TOM DAVIS AND MASON WOO, *OpenGL Programming Guide*, Addison Wesley, 1993

[23] OPENGL EXTENSION REGISTRY, *NV_depth_clamp*, 2003, available online: http://oss.sgi.com/projects/ogl-sample/registry/NV/depth_clamp.txt

[24] MATT OLSON AND HAO ZHANG, *Silhouette Extraction in Hough Space*, Eurographics Proceedings **25**, 2006, available online: http://www.cs.sfu.ca/~haoz/pubs/06_eg_hough.pdf#search=%22silhouette%20extraction%20hough%20space%22

[25] ASHU REGE, *Shadow Considerations*, available online: http://download.nvidia.com/developer/presentations/2004/6800_Leagues/6800_Leagues_Shadows.pdf

[26] LANCE WILLIAMS, *Casting Curved Shadows on Curved Surfaces*, Siggraph Proceedings **12**, 270–274, 1978, available online: http://accad.osu.edu/~waynec/history/PDFs/shadowmaps.pdf#search=%22casting%20curved%20surfaces%22

Günter Wallner
email: gw@autoteles.org

Department of Geometry
University of Applied Arts Vienna